KNOWNSEC

Degree of secrecy ★ | Project team open

Smart Contract Audit Report

Security status

Safe

★ ★ ★ ★ ★

Principal tester: Knownsec blockchain security team

Version Summary

Content	Date	Version
Editing Document	20210330	V1.0

Report Information

Title	Version	Document Number	Type
YouSwap Smart Contract Audit Report	V1.0		Open to project team

Copyright Notice

Knownsec only issues this report for facts that have occurred or existed before the issuance of this report, and assumes corresponding responsibilities for this. Knownsec is unable to determine the security status of its smart contracts and is not responsible for the facts that will occur or exist in the future. The security audit analysis and other content made in this report are only based on the documents and information provided to us by the information provider as of the time this report is issued. Knownsec's assumption: There is no missing, tampered, deleted or concealed information. If the information provided is missing, tampered with, deleted, concealed or reflected in the actual situation, Knownsec shall not be liable for any losses and adverse effects caused thereby.

Knownsec

Table of Contents

Knownsec

Knownsec

Knownsec

1. Introduction

The effective test time of this report is from From March 26, 2021 to March 27, 2021 . During this period, the security and standardization of **the smart contract code of the YouSwap** will be audited and used as the statistical basis for the report.

The scope of this smart contract security audit does not include external contract calls, new attack methods that may appear in the future, and code after contract upgrades or tampering. (With the development of the project, the smart contract may add a new pool , New functional modules, new external contract calls, etc.), does not include front-end security and server security.

In this audit report, engineers conducted a comprehensive analysis of the common vulnerabilities of smart contracts (Chapter 3). **The smart contract code of the YouSwap** is comprehensively assessed as **SAFE**.

Results of this smart contract security audit： SAFE

Since the testing is under non-production environment, all codes are the latest version. In addition, the testing process is communicated with the relevant engineer, and testing operations are carried out under the controllable operational risk to avoid production during the testing process, such as: Operational risk, code security risk.

Report information of this audit:

Report Number：

Report query address link:

Target information of the YouSwap audit:

Target information	
Token name	YouSwap
Contract address	YouswapFactoryV1： https://etherscan.io/address/0x0604F2781eF5712130Af4d941cb79257513d1693#code

Knownsec

Code type	Ethereum smart contract code
Code language	solidity

Contract documents and hash:

Contract documents	MD5
YouswapFactoryV1.sol	109624393D08BECD7737278580593182

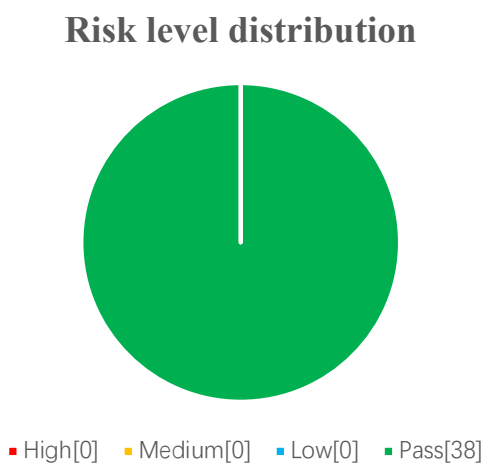
Knownsec

2. Code vulnerability analysis

2.1 Vulnerability Level Distribution

Vulnerability risk statistics by level：

Vulnerability risk level statistics table			
High	Medium	Low	Pass
0	0	0	38



Kno

2.2 Audit Result

Result of audit			
Audit Target	**Audit**	**Status**	**Audit Description**
Business security testing	**Invitation processing logic design**	Pass	After testing, there is no such safety vulnerability.
	User registration logic design	Pass	After testing, there is no such safety vulnerability.
	owner permission transfer	Pass	After testing, there is no such safety vulnerability.
	User pledge logic design	Pass	After testing, there is no such safety vulnerability.
	Reward extraction logic design	Pass	After testing, there is no such safety vulnerability.
	Calculation logic of computing power percentage	Pass	After testing, there is no such safety vulnerability.
	Logic design of pending rewards	Pass	After testing, there is no such safety vulnerability.
	Logic design of lower-level revenue contribution	Pass	After testing, there is no such safety vulnerability.
	Personal benefit bonus logic	Pass	After testing, there is no such safety vulnerability.
	New mining pool logic design	Pass	After testing, there is no such safety vulnerability.
	Mining pool information modification logic	Pass	After testing, there is no such safety vulnerability.

Basic code vulnerability detection	**Compiler version security**	Pass	After testing, there is no such safety vulnerability.
	Redundant code	Pass	After testing, there is no such safety vulnerability.
	Use of safe arithmetic library	Pass	After testing, there is no such safety vulnerability.
	Not recommended encoding	Pass	After testing, there is no such safety vulnerability.
	Reasonable use of require/assert	Pass	After testing, there is no such safety vulnerability.
	fallback function safety	Pass	After testing, there is no such safety vulnerability.
	tx.oriigin authentication	Pass	After testing, there is no such safety vulnerability.
	Owner permission control	Pass	After testing, there is no such safety vulnerability.
	Gas consumption detection	Pass	After testing, there is no such safety vulnerability.
	call injection attack	Pass	After testing, there is no such safety vulnerability.
	Low-level function safety	Pass	After testing, there is no such safety vulnerability.
	Vulnerability of additional token issuance	Pass	After testing, there is no such safety vulnerability.
	Access control defect detection	Pass	After testing, there is no such safety vulnerability.

	Numerical overflow detection	Pass	After testing, there is no such safety vulnerability.
	Arithmetic accuracy error	Pass	After testing, there is no such safety vulnerability.
	Wrong use of random number detection	Pass	After testing, there is no such safety vulnerability.
	Unsafe interface use	Pass	After testing, there is no such safety vulnerability.
	Variable coverage	Pass	After testing, there is no such safety vulnerability.
	Uninitialized storage pointer	Pass	After testing, there is no such safety vulnerability.
	Return value call verification	Pass	After testing, there is no such safety vulnerability.
	Transaction order dependency detection	Pass	After testing, there is no such safety vulnerability.
	Timestamp dependent attack	Pass	After testing, there is no such safety vulnerability.
	Denial of service attack detection	Pass	After testing, there is no such safety vulnerability.
	Fake recharge vulnerability detection	Pass	After testing, there is no such safety vulnerability.
	Reentry attack detection	Pass	After testing, there is no such safety vulnerability.
	Replay attack detection	Pass	After testing, there is no such safety vulnerability.
	Rearrangement attack detection	Pass	After testing, there is no such safety vulnerability.

Knownsec

3. Analysis of code audit results

3.1. Invitation processing logic design【PASS】

Audit the logic design of user invitation processing, check whether the parameters are verified, function caller authority verification, and whether the related logic design is reasonable.

Audit analysis: The relevant logic design is reasonable and correct.

```
function inviteCount() override external view returns (uint256) {
        return inviteUserInfoV1.length;
    }

    function inviteUpper1(address _owner) override external view returns (address) {
        return inviteUserInfoV2[_owner].upper;
    }

    function inviteUpper2(address _owner) override external view returns (address, address) {
        address upper1 = inviteUserInfoV2[_owner].upper;
        address upper2 = address(0);
        if (address(0) != upper1) {
            upper2 = inviteUserInfoV2[upper1].upper;
        }

        return (upper1, upper2);
    }

    function inviteLower1(address _owner) override external view returns (address[] memory) {
        return inviteUserInfoV2[_owner].lowers;
    }

    function inviteLower2(address _owner) override external view returns (address[] memory,
address[] memory) {
        address[] memory lowers1 = inviteUserInfoV2[_owner].lowers;
```

Knownsec

```
        uint256 count = 0;
        uint256 lowers1Len = lowers1.length;
        for (uint256 i = 0; i < lowers1Len; i++) {
            count += inviteUserInfoV2[lowers1[i]].lowers.length;
        }
        address[] memory lowers;
        address[] memory lowers2 = new address[](count);
        count = 0;
        for (uint256 i = 0; i < lowers1Len; i++) {
            lowers = inviteUserInfoV2[lowers1[i]].lowers;
            for (uint256 j = 0; j < lowers.length; j++) {
                lowers2[count] = lowers[j];
                count++;
            }
        }

        return (lowers1, lowers2);
    }

    function inviteLower2Count(address _owner) override external view returns (uint256, uint256) {
        address[] memory lowers1 = inviteUserInfoV2[_owner].lowers;
        uint256 lowers2Len = 0;
        uint256 len = lowers1.length;
        for (uint256 i = 0; i < len; i++) {
            lowers2Len += inviteUserInfoV2[lowers1[i]].lowers.length;
        }
        return (lowers1.length, lowers2Len);
    }
    function acceptInvitation(address _inviter) override external returns (bool) {
        require(msg.sender != _inviter, ErrorCode.FORBIDDEN);
        UserInfo storage user = inviteUserInfoV2[msg.sender];
        require(0 == user.startBlock, ErrorCode.REGISTERED);
        UserInfo storage upper = inviteUserInfoV2[_inviter];
```

Knownsec

```
        if (0 == upper.startBlock) {
            upper.upper = ZERO;
            upper.startBlock = block.number;
            inviteUserInfoV1.push(_inviter);

            emit InviteV1(_inviter, upper.upper, upper.startBlock);
        }
        user.upper = _inviter;
        upper.lowers.push(msg.sender);
        user.startBlock = block.number;
        inviteUserInfoV1.push(msg.sender);

        emit InviteV1(msg.sender, user.upper, user.startBlock);

        return true;
    }
    function acceptInvitation(address _inviter) override external returns (bool) {
        require(msg.sender != _inviter, ErrorCode.FORBIDDEN);
        UserInfo storage user = inviteUserInfoV2[msg.sender];
        require(0 == user.startBlock, ErrorCode.REGISTERED);
        UserInfo storage upper = inviteUserInfoV2[_inviter];
        if (0 == upper.startBlock) {
            upper.upper = ZERO;
            upper.startBlock = block.number;
            inviteUserInfoV1.push(_inviter);

            emit InviteV1(_inviter, upper.upper, upper.startBlock);
        }
        user.upper = _inviter;
        upper.lowers.push(msg.sender);
        user.startBlock = block.number;
        inviteUserInfoV1.push(msg.sender);
```

Knownsec

```
        emit InviteV1(msg.sender, user.upper, user.startBlock);

        return true;
    }
    function inviteBatch(address[] memory _invitees) override external returns (uint, uint) {
        uint len = _invitees.length;
        require(len <= 100, ErrorCode.PARAMETER_TOO_LONG);
        UserInfo storage user = inviteUserInfoV2[msg.sender];
        if (0 == user.startBlock) {
            user.upper = ZERO;
            user.startBlock = block.number;
            inviteUserInfoV1.push(msg.sender);

            emit InviteV1(msg.sender, user.upper, user.startBlock);
        }
        uint count = 0;
        for (uint i = 0; i < len; i++) {
            if ((address(0) != _invitees[i]) && (msg.sender != _invitees[i])) {
                UserInfo storage lower = inviteUserInfoV2[_invitees[i]];
                if (0 == lower.startBlock) {
                    lower.upper = msg.sender;
                    lower.startBlock = block.number;
                    user.lowers.push(_invitees[i]);
                    inviteUserInfoV1.push(_invitees[i]);
                    count++;

                    emit InviteV1(_invitees[i], msg.sender, lower.startBlock);
                }
            }
        }

        return (len, count);
    }
```

Knownsec

Recommendation： nothing.

3.2. User registration logic design【PASS】

Audit the logic design of user registration, check whether the parameters are verified, function caller authority verification, and related logic design is reasonable.

Audit analysis: The relevant logic design is reasonable and correct.

```
function register() override external returns (bool) {
        UserInfo storage user = inviteUserInfoV2[tx.origin];
        require(0 == user.startBlock, ErrorCode.REGISTERED);
        user.upper = ZERO;
        user.startBlock = block.number;
        inviteUserInfoV1.push(tx.origin);
        emit InviteV1(tx.origin, user.upper, user.startBlock);

        return true;
    }
```

Recommendation： nothing.

3.3. Ownership transfer【PASS】

Audit the logic design of contract owner authority transfer, check whether there is verification of parameters, function caller authority verification, and whether the related logic design is reasonable.

Audit analysis: The relevant logic design is reasonable and correct.

```
function transferOwnership(address _owner) override external {
        require(owner == msg.sender, ErrorCode.FORBIDDEN);
        require((address(0) != _owner) && (owner != _owner),
ErrorCode.INVALID_ADDRESSES);
        _setOperateOwner(owner, false);
```

Knownsec

```
        _setOperateOwner(_owner, true);
        owner = _owner;
    }
        function _setOperateOwner(address _address, bool _bool) internal {
        require(owner == msg.sender, ErrorCode.FORBIDDEN);
        operateOwner[_address] = _bool;
    }
```

Recommendation： nothing.

3.4. User commitment logic design【PASS】

Audit the pledge logic design, check whether the parameters are verified, the function caller's permission verification, and the related logic design is reasonable.

Audit analysis: The relevant logic design is reasonable and correct.

```
 function deposit(uint256 _pool, uint256 _amount) override external {
        require(0 < _amount, ErrorCode.FORBIDDEN);
        PoolInfo storage poolInfo = poolInfos[_pool];
        require((address(0) != poolInfo.lp) && (poolInfo.startBlock <= block.number),
ErrorCode.MINING_NOT_STARTED);
        //require(0 == poolInfo.endBlock, ErrorCode.END_OF_MINING);
        (, uint256 startBlock) = invite.inviteUserInfoV2(msg.sender);
        if (0 == startBlock) {
            invite.register();
            emit InviteRegister(msg.sender);
        }
        IERC20(poolInfo.lp).safeTransferFrom(msg.sender, address(this), _amount);

        (address upper1, address upper2) = invite.inviteUpper2(msg.sender);
        computeReward(_pool);
        provideReward(_pool, poolInfo.rewardPerShare, poolInfo.lp, msg.sender, upper1, upper2);
        addPower(_pool, msg.sender, _amount, upper1, upper2);
```

Knownsec

```
        setRewardDebt(_pool, poolInfo.rewardPerShare, msg.sender, upper1, upper2);
        emit Stake(_pool, poolInfo.lp, msg.sender, _amount);
    }
    function computeReward(uint256 _pool) internal {
        PoolInfo storage poolInfo = poolInfos[_pool];
        if ((0 < poolInfo.totalPower) && (poolInfo.rewardProvide < poolInfo.rewardTotal)) {
            uint256 reward = (block.number -
poolInfo.lastRewardBlock).mul(poolInfo.rewardPerBlock);
            if (poolInfo.rewardProvide.add(reward) > poolInfo.rewardTotal) {
                reward = poolInfo.rewardTotal.sub(poolInfo.rewardProvide);
                poolInfo.endBlock = block.number;
            }

            rewardTotal = rewardTotal.add(reward);
            poolInfo.rewardProvide = poolInfo.rewardProvide.add(reward);
            poolInfo.rewardPerShare =
poolInfo.rewardPerShare.add(reward.mul(1e24).div(poolInfo.totalPower));
            poolInfo.lastRewardBlock = block.number;

            emit Mint(_pool, poolInfo.lp, reward);

            if (0 < poolInfo.endBlock) {
                emit EndPool(_pool, poolInfo.lp);
            }
        }
    }
    function provideReward(uint256 _pool, uint256 _rewardPerShare, address _lp, address _user,
address _upper1, address _upper2) internal {
        uint256 inviteReward = 0;
        uint256 pledgeReward = 0;
        UserInfo storage userInfo = pledgeUserInfo[_pool][_user];
        if ((0 < userInfo.invitePower) || (0 < userInfo.pledgePower)) {
```

Knownsec

```
                inviteReward =
userInfo.invitePower.mul(_rewardPerShare).sub(userInfo.inviteRewardDebt).div(1e24);
                pledgeReward =
userInfo.pledgePower.mul(_rewardPerShare).sub(userInfo.pledgeRewardDebt).div(1e24);

                userInfo.pendingReward =
userInfo.pendingReward.add(inviteReward.add(pledgeReward));

                RewardInfo storage userRewardInfo = rewardInfos[_user];
                userRewardInfo.inviteReward = userRewardInfo.inviteReward.add(inviteReward);
                userRewardInfo.pledgeReward = userRewardInfo.pledgeReward.add(pledgeReward);
            }

            if (0 < userInfo.pendingReward) {
                you.mint(_user, userInfo.pendingReward);

                RewardInfo storage userRewardInfo = rewardInfos[_user];
                userRewardInfo.receiveReward = userRewardInfo.inviteReward;

                emit WithdrawReward(_pool, _lp, _user, userInfo.pendingReward);

                userInfo.pendingReward = 0;
            }

            if (address(0) != _upper1) {
                UserInfo storage upper1Info = pledgeUserInfo[_pool][_upper1];
                if ((0 < upper1Info.invitePower) || (0 < upper1Info.pledgePower)) {
                    inviteReward =
upper1Info.invitePower.mul(_rewardPerShare).sub(upper1Info.inviteRewardDebt).div(1e24);
                    pledgeReward =
upper1Info.pledgePower.mul(_rewardPerShare).sub(upper1Info.pledgeRewardDebt).div(1e24);
```

Knownsec

```
                upper1Info.pendingReward =
upper1Info.pendingReward.add(inviteReward.add(pledgeReward));

                RewardInfo storage upper1RewardInfo = rewardInfos[_upper1];
                upper1RewardInfo.inviteReward =
upper1RewardInfo.inviteReward.add(inviteReward);
                upper1RewardInfo.pledgeReward =
upper1RewardInfo.pledgeReward.add(pledgeReward);
            }

            if (address(0) != _upper2) {
                UserInfo storage upper2Info = pledgeUserInfo[_pool][_upper2];
                if ((0 < upper2Info.invitePower) || (0 < upper2Info.pledgePower)) {
                    inviteReward =
upper2Info.invitePower.mul(_rewardPerShare).sub(upper2Info.inviteRewardDebt).div(1e24);
                    pledgeReward =
upper2Info.pledgePower.mul(_rewardPerShare).sub(upper2Info.pledgeRewardDebt).div(1e24);

                    upper2Info.pendingReward =
upper2Info.pendingReward.add(inviteReward.add(pledgeReward));

                    RewardInfo storage upper2RewardInfo = rewardInfos[_upper2];
                    upper2RewardInfo.inviteReward =
upper2RewardInfo.inviteReward.add(inviteReward);
                    upper2RewardInfo.pledgeReward =
upper2RewardInfo.pledgeReward.add(pledgeReward);
                }
            }
        }
    }
    function addPower(uint256 _pool, address _user, uint256 _amount, address _upper1, address
_upper2) internal {
        PoolInfo storage poolInfo = poolInfos[_pool];
```

Knownsec

```
poolInfo.amount = poolInfo.amount.add(_amount);

uint256 pledgePower = _amount;
UserInfo storage userInfo = pledgeUserInfo[_pool][_user];
userInfo.amount = userInfo.amount.add(_amount);
userInfo.pledgePower = userInfo.pledgePower.add(pledgePower);
poolInfo.totalPower = poolInfo.totalPower.add(pledgePower);
if (0 == userInfo.startBlock) {
    userInfo.startBlock = block.number;
    pledgeAddresss[_pool].push(msg.sender);
}

uint256 upper1InvitePower = 0;
uint256 upper2InvitePower = 0;

if (address(0) != _upper1) {
    uint256 inviteSelfPower = pledgePower.mul(inviteSelfReward).div(100);
    userInfo.invitePower = userInfo.invitePower.add(inviteSelfPower);
    poolInfo.totalPower = poolInfo.totalPower.add(inviteSelfPower);

    uint256 invite1Power = pledgePower.mul(invite1Reward).div(100);
    UserInfo storage upper1Info = pledgeUserInfo[_pool][_upper1];
    upper1Info.invitePower = upper1Info.invitePower.add(invite1Power);
    upper1InvitePower = upper1Info.invitePower;
    poolInfo.totalPower = poolInfo.totalPower.add(invite1Power);
    if (0 == upper1Info.startBlock) {
        upper1Info.startBlock = block.number;
        pledgeAddresss[_pool].push(_upper1);
    }
}

if (address(0) != _upper2) {
    uint256 invite2Power = pledgePower.mul(invite2Reward).div(100);
```

Knownsec

```
            UserInfo storage upper2Info = pledgeUserInfo[_pool][_upper2];
            upper2Info.invitePower = upper2Info.invitePower.add(invite2Power);
            upper2InvitePower = upper2Info.invitePower;
            poolInfo.totalPower = poolInfo.totalPower.add(invite2Power);
            if (0 == upper2Info.startBlock) {
                upper2Info.startBlock = block.number;
                pledgeAddresss[_pool].push(_upper2);
            }
        }

        emit UpdatePower(_pool, poolInfo.lp, poolInfo.totalPower, _user, userInfo.invitePower,
userInfo.pledgePower, _upper1, upper1InvitePower, _upper2, upper2InvitePower);
    }
    function setRewardDebt(uint256 _pool, uint256 _rewardPerShare, address _user, address
_upper1, address _upper2) internal {
        UserInfo storage userInfo = pledgeUserInfo[_pool][_user];
        userInfo.inviteRewardDebt = userInfo.invitePower.mul(_rewardPerShare);
        userInfo.pledgeRewardDebt = userInfo.pledgePower.mul(_rewardPerShare);

        if (address(0) != _upper1) {
            UserInfo storage upper1Info = pledgeUserInfo[_pool][_upper1];
            upper1Info.inviteRewardDebt = upper1Info.invitePower.mul(_rewardPerShare);
            upper1Info.pledgeRewardDebt = upper1Info.pledgePower.mul(_rewardPerShare);

            if (address(0) != _upper2) {
                UserInfo storage upper2Info = pledgeUserInfo[_pool][_upper2];
                upper2Info.inviteRewardDebt = upper2Info.invitePower.mul(_rewardPerShare);
                upper2Info.pledgeRewardDebt = upper2Info.pledgePower.mul(_rewardPerShare);
            }
        }
    }
```

Recommendation： nothing.

Knownsec

3.5. Reward extraction logic design【PASS】

Audit the design of the withdrawal logic, check whether the parameters are verified, the function caller's permission verification, and whether the related logic design is reasonable.

Audit analysis: The relevant logic design is reasonable and correct.

```
    function withdraw(uint256 _pool, uint256 _amount) override external {
        PoolInfo storage poolInfo = poolInfos[_pool];
        require((address(0) != poolInfo.lp) && (poolInfo.startBlock <= block.number),
ErrorCode.MINING_NOT_STARTED);
        if (0 < _amount) {
            UserInfo storage userInfo = pledgeUserInfo[_pool][msg.sender];
            require(_amount <= userInfo.amount, ErrorCode.BALANCE_INSUFFICIENT);
            IERC20(poolInfo.lp).safeTransfer(msg.sender, _amount);

            emit UnStake(_pool, poolInfo.lp, msg.sender, _amount);
        }
        (address _upper1, address _upper2) = invite.inviteUpper2(msg.sender);
        computeReward(_pool);
        provideReward(_pool, poolInfo.rewardPerShare, poolInfo.lp, msg.sender, _upper1,
_upper2);

        if (0 < _amount) {
            subPower(_pool, msg.sender, _amount, _upper1, _upper2);
        }

        setRewardDebt(_pool, poolInfo.rewardPerShare, msg.sender, _upper1, _upper2);
    }
    function computeReward(uint256 _pool) internal {
        PoolInfo storage poolInfo = poolInfos[_pool];
        if ((0 < poolInfo.totalPower) && (poolInfo.rewardProvide < poolInfo.rewardTotal)) {
            uint256 reward = (block.number -
poolInfo.lastRewardBlock).mul(poolInfo.rewardPerBlock);
```

Knownsec

```
                if (poolInfo.rewardProvide.add(reward) > poolInfo.rewardTotal) {
                    reward = poolInfo.rewardTotal.sub(poolInfo.rewardProvide);
                    poolInfo.endBlock = block.number;
                }

                rewardTotal = rewardTotal.add(reward);
                poolInfo.rewardProvide = poolInfo.rewardProvide.add(reward);
                poolInfo.rewardPerShare =
poolInfo.rewardPerShare.add(reward.mul(1e24).div(poolInfo.totalPower));
                poolInfo.lastRewardBlock = block.number;

                emit Mint(_pool, poolInfo.lp, reward);

                if (0 < poolInfo.endBlock) {
                    emit EndPool(_pool, poolInfo.lp);
                }
            }
        }
        function provideReward(uint256 _pool, uint256 _rewardPerShare, address _lp, address _user,
address _upper1, address _upper2) internal {
            uint256 inviteReward = 0;
            uint256 pledgeReward = 0;
            UserInfo storage userInfo = pledgeUserInfo[_pool][_user];
            if ((0 < userInfo.invitePower) || (0 < userInfo.pledgePower)) {
                inviteReward =
userInfo.invitePower.mul(_rewardPerShare).sub(userInfo.inviteRewardDebt).div(1e24);
                pledgeReward =
userInfo.pledgePower.mul(_rewardPerShare).sub(userInfo.pledgeRewardDebt).div(1e24);

                userInfo.pendingReward =
userInfo.pendingReward.add(inviteReward.add(pledgeReward));

                RewardInfo storage userRewardInfo = rewardInfos[_user];
```

Knownsec

```
            userRewardInfo.inviteReward = userRewardInfo.inviteReward.add(inviteReward);
            userRewardInfo.pledgeReward = userRewardInfo.pledgeReward.add(pledgeReward);
        }

        if (0 < userInfo.pendingReward) {
            you.mint(_user, userInfo.pendingReward);

            RewardInfo storage userRewardInfo = rewardInfos[_user];
            userRewardInfo.receiveReward = userRewardInfo.inviteReward;

            emit WithdrawReward(_pool, _lp, _user, userInfo.pendingReward);

            userInfo.pendingReward = 0;
        }

        if (address(0) != _upper1) {
            UserInfo storage upper1Info = pledgeUserInfo[_pool][_upper1];
            if ((0 < upper1Info.invitePower) || (0 < upper1Info.pledgePower)) {
                inviteReward =
upper1Info.invitePower.mul(_rewardPerShare).sub(upper1Info.inviteRewardDebt).div(1e24);
                pledgeReward =
upper1Info.pledgePower.mul(_rewardPerShare).sub(upper1Info.pledgeRewardDebt).div(1e24);

                upper1Info.pendingReward =
upper1Info.pendingReward.add(inviteReward.add(pledgeReward));

                RewardInfo storage upper1RewardInfo = rewardInfos[_upper1];
                upper1RewardInfo.inviteReward =
upper1RewardInfo.inviteReward.add(inviteReward);
                upper1RewardInfo.pledgeReward =
upper1RewardInfo.pledgeReward.add(pledgeReward);
            }
```

Knownsec

```
                if (address(0) != _upper2) {
                    UserInfo storage upper2Info = pledgeUserInfo[_pool][_upper2];
                    if ((0 < upper2Info.invitePower) || (0 < upper2Info.pledgePower)) {
                        inviteReward =
upper2Info.invitePower.mul(_rewardPerShare).sub(upper2Info.inviteRewardDebt).div(1e24);
                        pledgeReward =
upper2Info.pledgePower.mul(_rewardPerShare).sub(upper2Info.pledgeRewardDebt).div(1e24);

                        upper2Info.pendingReward =
upper2Info.pendingReward.add(inviteReward.add(pledgeReward));

                        RewardInfo storage upper2RewardInfo = rewardInfos[_upper2];
                        upper2RewardInfo.inviteReward =
upper2RewardInfo.inviteReward.add(inviteReward);
                        upper2RewardInfo.pledgeReward =
upper2RewardInfo.pledgeReward.add(pledgeReward);
                    }
                }
            }
        }
    function subPower(uint256 _pool, address _user, uint256 _amount, address _upper1, address
_upper2) internal {
        PoolInfo storage poolInfo = poolInfos[_pool];
        UserInfo storage userInfo = pledgeUserInfo[_pool][_user];
        poolInfo.amount = poolInfo.amount.sub(_amount);

        uint256 pledgePower = _amount;
        userInfo.amount = userInfo.amount.sub(_amount);
        userInfo.pledgePower = userInfo.pledgePower.sub(pledgePower);
        poolInfo.totalPower = poolInfo.totalPower.sub(pledgePower);

        uint256 upper1InvitePower = 0;
        uint256 upper2InvitePower = 0;
```

Knownsec

```
        if (address(0) != _upper1) {
            uint256 inviteSelfPower = pledgePower.mul(inviteSelfReward).div(100);
            userInfo.invitePower = userInfo.invitePower.sub(inviteSelfPower);
            poolInfo.totalPower = poolInfo.totalPower.sub(inviteSelfPower);

            UserInfo storage upper1Info = pledgeUserInfo[_pool][_upper1];
            if (0 < upper1Info.startBlock) {
                uint256 invite1Power = pledgePower.mul(invite1Reward).div(100);
                upper1Info.invitePower = upper1Info.invitePower.sub(invite1Power);
                upper1InvitePower = upper1Info.invitePower;
                poolInfo.totalPower = poolInfo.totalPower.sub(invite1Power);

                if (address(0) != _upper2) {
                    UserInfo storage upper2Info = pledgeUserInfo[_pool][_upper2];
                    if (0 < upper2Info.startBlock) {
                        uint256 invite2Power = pledgePower.mul(invite2Reward).div(100);
                        upper2Info.invitePower = upper2Info.invitePower.sub(invite2Power);
                        upper2InvitePower = upper2Info.invitePower;
                        poolInfo.totalPower = poolInfo.totalPower.sub(invite2Power);
                    }
                }
            }
        }

        emit UpdatePower(_pool, poolInfo.lp, poolInfo.totalPower, _user, userInfo.invitePower,
userInfo.pledgePower, _upper1, upper1InvitePower, _upper2, upper2InvitePower);
    }
    function setRewardDebt(uint256 _pool, uint256 _rewardPerShare, address _user, address
_upper1, address _upper2) internal {
        UserInfo storage userInfo = pledgeUserInfo[_pool][_user];
        userInfo.inviteRewardDebt = userInfo.invitePower.mul(_rewardPerShare);
        userInfo.pledgeRewardDebt = userInfo.pledgePower.mul(_rewardPerShare);
```

Knownsec

```
        if (address(0) != _upper1) {
            UserInfo storage upper1Info = pledgeUserInfo[_pool][_upper1];
            upper1Info.inviteRewardDebt = upper1Info.invitePower.mul(_rewardPerShare);
            upper1Info.pledgeRewardDebt = upper1Info.pledgePower.mul(_rewardPerShare);

            if (address(0) != _upper2) {
                UserInfo storage upper2Info = pledgeUserInfo[_pool][_upper2];
                upper2Info.inviteRewardDebt = upper2Info.invitePower.mul(_rewardPerShare);
                upper2Info.pledgeRewardDebt = upper2Info.pledgePower.mul(_rewardPerShare);
            }
        }
    }
```

Recommendation： nothing.

3.6. Calculating logic for percentage of computing power 【PASS】

Audit the logic design of the powerScale computing power ratio calculation, check whether the parameters are verified, the function caller permission verification, and the related logic design is reasonable.

Audit analysis: The relevant logic design is reasonable and correct.

```
    function powerScale(uint256 _pool, address _user) override external view returns (uint256) {
        PoolInfo memory poolInfo = poolInfos[_pool];
        if (0 == poolInfo.totalPower) {
            return 0;
        }

        UserInfo memory userInfo = pledgeUserInfo[_pool][_user];
        return (userInfo.invitePower.add(userInfo.pledgePower).mul(100)).div(poolInfo.totalPower);
    }
```

Knownsec

Recommendation： nothing.

3.7. Logical design of pending rewards【PASS】

Audit the logic design of pendingReward to check whether the parameters are verified, the permission of the function caller is verified, and whether the related logic design is reasonable.

Audit analysis: The relevant logic design is reasonable and correct.

```
function pendingReward(uint256 _pool, address _user) override external view returns (uint256) {
        uint256 totalReward = 0;
        PoolInfo memory poolInfo = poolInfos[_pool];
        if (address(0) != poolInfo.lp && (poolInfo.startBlock <= block.number)) {
            uint256 rewardPerShare = 0;
            if (0 < poolInfo.totalPower) {
                uint256 reward = (block.number -
poolInfo.lastRewardBlock).mul(poolInfo.rewardPerBlock);
                if (poolInfo.rewardProvide.add(reward) > poolInfo.rewardTotal) {
                    reward = poolInfo.rewardTotal.sub(poolInfo.rewardProvide);
                }
                rewardPerShare = reward.mul(1e24).div(poolInfo.totalPower);
            }
            rewardPerShare = rewardPerShare.add(poolInfo.rewardPerShare);

            UserInfo memory userInfo = pledgeUserInfo[_pool][_user];
            totalReward = userInfo.pendingReward;
            totalReward =
totalReward.add(userInfo.invitePower.mul(rewardPerShare).sub(userInfo.inviteRewardDebt).div(1e24
));
            totalReward =
totalReward.add(userInfo.pledgePower.mul(rewardPerShare).sub(userInfo.pledgeRewardDebt).div(1e2
4));
```

Knownsec

```
        }

        return totalReward;
    }
```

Recommendation： nothing.

3.8. Logical design of lower income contribution【PASS】

Audit the logic design of the lower-level revenue contribution to check whether the parameters are verified, the function caller's permission verification, and whether the related logic design is reasonable.

Audit analysis: The relevant logic design is reasonable and correct.

```
function rewardContribute(address _user, address _lower) override external view returns (uint256) {
        if ((address(0) == _user) || (address(0) == _lower)) {
            return 0;
        }

        uint256 inviteReward = 0;
        (address upper1, address upper2) = invite.inviteUpper2(_lower);
        if (_user == upper1) {
            inviteReward = rewardInfos[_lower].pledgeReward.mul(invite1Reward).div(100);
        }else if (_user == upper2) {
            inviteReward = rewardInfos[_lower].pledgeReward.mul(invite2Reward).div(100);
        }

        return inviteReward;
    }
```

Recommendation： nothing.

Knownsec

3.9. Personal benefit bonus logic【PASS】

Audit the logic design of personal income bonus, check whether there is verification of parameters, function caller authority verification, and whether the related logic design is reasonable.

Audit analysis: The relevant logic design is reasonable and correct.

```
function selfReward(address _user) override external view returns (uint256) {
        address upper1 = invite.inviteUpper1(_user);
        if (address(0) == upper1) {
            return 0;
        }
        RewardInfo memory userRewardInfo = rewardInfos[_user];
        return userRewardInfo.pledgeReward.mul(inviteSelfReward).div(100);
    }
```

Recommendation： nothing.

3.10. New mining pool logic design【PASS】

Audit the logic design of the new mining pool, check whether the parameters are verified, the function caller's permission verification, and whether the related logic design is reasonable.

Audit analysis: The relevant logic design is reasonable and correct.

```
function addPool(string memory _name, address _lp, uint256 _startBlock, uint256 _rewardTotal)
override external returns (bool) {
        require(operateOwner[msg.sender] && (address(0) != _lp) && (address(this) != _lp),
ErrorCode.FORBIDDEN);
        _startBlock = _startBlock < block.number ? block.number : _startBlock;
        uint256 _pool = poolCount;
        poolCount = poolCount.add(1);

        PoolViewInfo storage poolViewInfo = poolViewInfos[_pool];
```

Knownsec

```
        poolViewInfo.lp = _lp;
        poolViewInfo.name = _name;
        poolViewInfo.multiple = 1;
        poolViewInfo.priority = _pool.mul(100);

        PoolInfo storage poolInfo = poolInfos[_pool];
        poolInfo.startBlock = _startBlock;
        poolInfo.rewardTotal = _rewardTotal;
        poolInfo.rewardProvide = 0;
        poolInfo.lp = _lp;
        poolInfo.amount = 0;
        poolInfo.lastRewardBlock = _startBlock.sub(1);
        poolInfo.rewardPerBlock = rewardPerBlock;
        poolInfo.totalPower = 0;
        poolInfo.endBlock = 0;
        poolInfo.rewardPerShare = 0;

        emit UpdatePool(true, _pool, poolInfo.lp, poolViewInfo.name, poolInfo.startBlock,
poolInfo.rewardTotal, poolInfo.rewardPerBlock, poolViewInfo.multiple, poolViewInfo.priority);

        return true;
    }
```

Recommendation： nothing.

3.11. Mining pool information modification logic【PASS】

Audit the logic design of the mining pool information modification, check whether the parameters are verified, the function caller permission verification, and whether the related logic design is reasonable.

Audit analysis: The relevant logic design is reasonable and correct.

```
function setRewardPerBlock(uint256 _pool, uint256 _rewardPerBlock) override external {
```

Knownsec

```
        require(operateOwner[msg.sender], ErrorCode.FORBIDDEN);
        PoolInfo storage poolInfo = poolInfos[_pool];
        require((address(0) != poolInfo.lp) && (0 == poolInfo.endBlock),
ErrorCode.POOL_NOT_EXIST_OR_END_OF_MINING);
        poolInfo.rewardPerBlock = _rewardPerBlock;

        PoolViewInfo memory poolViewInfo = poolViewInfos[_pool];

        emit UpdatePool(false, _pool, poolInfo.lp, poolViewInfo.name, poolInfo.startBlock,
poolInfo.rewardTotal, poolInfo.rewardPerBlock, poolViewInfo.multiple, poolViewInfo.priority);
    }

    function setRewardTotal(uint256 _pool, uint256 _rewardTotal) override external {
        require(operateOwner[msg.sender], ErrorCode.FORBIDDEN);
        PoolInfo storage poolInfo = poolInfos[_pool];
        require((address(0) != poolInfo.lp) && (0 == poolInfo.endBlock),
ErrorCode.POOL_NOT_EXIST_OR_END_OF_MINING);
        require(poolInfo.rewardProvide < _rewardTotal,
ErrorCode.REWARDTOTAL_LESS_THAN_REWARDPROVIDE);
        poolInfo.rewardTotal = _rewardTotal;

        PoolViewInfo memory poolViewInfo = poolViewInfos[_pool];

        emit UpdatePool(false, _pool, poolInfo.lp, poolViewInfo.name, poolInfo.startBlock,
poolInfo.rewardTotal, poolInfo.rewardPerBlock, poolViewInfo.multiple, poolViewInfo.priority);
    }

    function setName(uint256 _pool, string memory _name) override external {
        require(operateOwner[msg.sender], ErrorCode.FORBIDDEN);
        PoolViewInfo storage poolViewInfo = poolViewInfos[_pool];
        require(address(0) != poolViewInfo.lp,
ErrorCode.POOL_NOT_EXIST_OR_END_OF_MINING);
        poolViewInfo.name = _name;
```

Knownsec

```
        PoolInfo memory poolInfo = poolInfos[_pool];

        emit UpdatePool(false, _pool, poolInfo.lp, poolViewInfo.name, poolInfo.startBlock,
poolInfo.rewardTotal, poolInfo.rewardPerBlock, poolViewInfo.multiple, poolViewInfo.priority);
    }

    function setMultiple(uint256 _pool, uint256 _multiple) override external {
        require(operateOwner[msg.sender], ErrorCode.FORBIDDEN);
        PoolViewInfo storage poolViewInfo = poolViewInfos[_pool];
        require(address(0) != poolViewInfo.lp,
ErrorCode.POOL_NOT_EXIST_OR_END_OF_MINING);
        poolViewInfo.multiple = _multiple;

        PoolInfo memory poolInfo = poolInfos[_pool];

        emit UpdatePool(false, _pool, poolInfo.lp, poolViewInfo.name, poolInfo.startBlock,
poolInfo.rewardTotal, poolInfo.rewardPerBlock, poolViewInfo.multiple, poolViewInfo.priority);
    }

    function setPriority(uint256 _pool, uint256 _priority) override external {
        require(operateOwner[msg.sender], ErrorCode.FORBIDDEN);
        PoolViewInfo storage poolViewInfo = poolViewInfos[_pool];
        require(address(0) != poolViewInfo.lp,
ErrorCode.POOL_NOT_EXIST_OR_END_OF_MINING);
        poolViewInfo.priority = _priority;

        PoolInfo memory poolInfo = poolInfos[_pool];

        emit UpdatePool(false, _pool, poolInfo.lp, poolViewInfo.name, poolInfo.startBlock,
poolInfo.rewardTotal, poolInfo.rewardPerBlock, poolViewInfo.multiple, poolViewInfo.priority);
    }
```

Knownsec

Recommendation： nothing.

Knownsec

4. Basic code vulnerability detection

4.1. Compiler version security【PASS】

Check whether a safe compiler version is used in the contract code implementation.

Audit result: After testing, the IDO contract compiler version is specified in the smart contract code as 0.5.16, TokenYou

The contract compiler version is above 0.6.0, and this security issue does not exist.

Recommendation： nothing.

4.2. Redundant code【PASS】

Check whether the contract code implementation contains redundant code.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.3. Use of safe arithmetic library【PASS】

Check whether the SafeMath safe arithmetic library is used in the contract code implementation.

Audit result: After testing, the SafeMath safe arithmetic library has been used in the smart contract code, and there is no such security problem.

Knownsec

Recommendation： nothing.

4.4. Not recommended encoding【PASS】

Check whether there is an encoding method that is not officially recommended or abandoned in the contract code implementation

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.5. Reasonable use of require/assert【PASS】

Check the rationality of the use of require and assert statements in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.6. Fallback function safety【PASS】

Check whether the fallback function is used correctly in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

Knownsec

4.7. tx.origin authentication【PASS】

tx.origin is a global variable of Solidity that traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in a smart contract makes the contract vulnerable to attacks like phishing.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.8. Owner permission control【PASS】

Check whether the owner in the contract code implementation has excessive authority. For example, arbitrarily modify other account balances, etc.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.9. Gas consumption detection【PASS】

Check whether the consumption of gas exceeds the maximum block limit.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

Knownsec

4.10. call injection attack【PASS】

When the call function is called, strict permission control should be done, or the function called by the call should be written dead.

Audit result: After testing, the smart contract does not use the call function, and this vulnerability does not exist.

Recommendation： nothing.

4.11. Low-level function safety【PASS】

Check whether there are security vulnerabilities in the use of low-level functions (call/delegatecall) in the contract code implementation

The execution context of the call function is in the called contract; the execution context of the delegatecall function is in the contract that currently calls the function.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.12. Vulnerability of additional token issuance【PASS】

Check whether there is a function that may increase the total amount of tokens in the token contract after initializing the total amount of tokens.

Knownsec

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.13. Access control defect detection【PASS】

Different functions in the contract should set reasonable permissions.

Check whether each function in the contract correctly uses keywords such as public and private for visibility modification, check whether the contract is correctly defined and use modifier to restrict access to key functions to avoid problems caused by unauthorized access.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.14. Numerical overflow detection【PASS】

The arithmetic problems in smart contracts refer to integer overflow and integer underflow.

Solidity can handle up to 256-bit numbers (2^256-1). If the maximum number increases by 1, it will overflow to 0. Similarly, when the number is an unsigned type, 0 minus 1 will underflow to get the maximum digital value.

Integer overflow and underflow are not a new type of vulnerability, but they are especially dangerous in smart contracts. Overflow conditions can lead to incorrect

Knownsec

results, especially if the possibility is not expected, which may affect the reliability and safety of the program.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.15. Arithmetic accuracy error【PASS】

As a programming language, Solidity has data structure design similar to ordinary programming languages, such as variables, constants, functions, arrays, functions, structures, etc. There is also a big difference between Solidity and ordinary programming languages-Solidity does not float Point type, and all the numerical calculation results of Solidity will only be integers, there will be no decimals, and it is not allowed to define decimal type data. Numerical calculations in the contract are indispensable, and the design of numerical calculations may cause relative errors. For example, the same level of calculations: 5/2*10=20, and 5*10/2=25, resulting in errors, which are larger in data The error will be larger and more obvious.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

Knownsec

4.16. Incorrect use of random numbers【PASS】

Smart contracts may need to use random numbers. Although the functions and variables provided by Solidity can access values that are obviously unpredictable, such as block.number and block.timestamp, they are usually more public than they appear or are affected by miners. These random numbers are predictable to a certain extent, so malicious users can usually copy it and rely on its unpredictability to attack the function.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.17. Unsafe interface usage【PASS】

Check whether unsafe interfaces are used in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.18. Variable coverage【PASS】

Check whether there are security issues caused by variable coverage in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Knownsec

Recommendation： nothing.

4.19. Uninitialized storage pointer【PASS】

In solidity, a special data structure is allowed to be a struct structure, and the local variables in the function are stored in storage or memory by default.

The existence of storage (memory) and memory (memory) are two different concepts. Solidity allows pointers to point to an uninitialized reference, while uninitialized local storage will cause variables to point to other storage variables, leading to variable coverage, or even more serious As a consequence, you should avoid initializing struct variables in functions during development.

Audit result: After testing, the smart contract code does not use structure, there is no such problem.

Recommendation： nothing.

4.20. Return value call verification【PASS】

This problem mostly occurs in smart contracts related to currency transfer, so it is also called silent failed delivery or unchecked delivery.

In Solidity, there are transfer(), send(), call.value() and other currency transfer methods, which can all be used to send ETH to an address. The difference is: When the transfer fails, it will be thrown and the state will be rolled back; Only 2300gas will be passed for calling to prevent reentry attacks; false will be returned when send fails; only 2300gas will be passed for calling to prevent reentry attacks; false will be

Knownsec

returned when call.value fails to be sent; all available gas will be passed for calling (can be Limit by passing in gas_value parameters), which cannot effectively prevent reentry attacks.

If the return value of the above send and call.value transfer functions is not checked in the code, the contract will continue to execute the following code, which may lead to unexpected results due to ETH sending failure.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.21. Transaction order dependency【PASS】

Since miners always get gas fees through codes that represent externally owned addresses (EOA), users can specify higher fees for faster transactions. Since the Ethereum blockchain is public, everyone can see the content of other people's pending transactions. This means that if a user submits a valuable solution, a malicious user can steal the solution and copy its transaction at a higher fee to preempt the original solution.

Audit result :After testing, the security problem does not exist in the smart contract code.

Knownsec

4.22. Timestamp dependency attack【PASS】

The timestamp of the data block usually uses the local time of the miner, and this time can fluctuate in the range of about 900 seconds. When other nodes accept a new block, it only needs to verify whether the timestamp is later than the previous block and The error with local time is within 900 seconds. A miner can profit from it by setting the timestamp of the block to satisfy the conditions that are beneficial to him as much as possible.

Check whether there are key functions that depend on the timestamp in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.23. Denial of service attack【PASS】

In the world of Ethereum, denial of service is fatal, and a smart contract that has suffered this type of attack may never be able to return to its normal working state. There may be many reasons for the denial of service of the smart contract, including malicious behavior as the transaction recipient, artificially increasing the gas required for computing functions to cause gas exhaustion, abusing access control to access the private component of the smart contract, using confusion and negligence, etc. Wait.

Knownsec

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.24. Fake recharge vulnerability【PASS】

The transfer function of the token contract uses the if judgment method to check the balance of the transfer initiator (msg.sender). When balances[msg.sender] <value, enter the else logic part and return false, and finally no exception is thrown. We believe that only if/else this kind of gentle judgment method is an imprecise coding method in sensitive function scenarios such as transfer.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.25. Reentry attack detection【PASS】

The **call.value()** function in Solidity consumes all the gas it receives when it is used to send ETH. When the **call.value()** function to send ETH occurs before the actual reduction of the sender's account balance, There is a risk of reentry attacks.

Audit results： After auditing, the vulnerability does not exist in the smart contract code.

Recommendation： nothing.

Knownsec

4.26. Replay attack detection【PASS】

If the contract involves the need for entrusted management, attention should be paid to the non-reusability of verification to avoid replay attacks

In the asset management system, there are often cases of entrusted management. The principal assigns assets to the trustee for management, and the principal pays a certain fee to the trustee. This business scenario is also common in smart contracts.

Audit results： After testing, the smart contract does not use the call function, and this vulnerability does not exist.

Recommendation： nothing.

4.27. Rearrangement attack detection【PASS】

A rearrangement attack refers to a miner or other party trying to "compete" with smart contract participants by inserting their own information into a list or mapping, so that the attacker has the opportunity to store their own information in the contract. in.

Audit results： After auditing, the vulnerability does not exist in the smart contract code.

Recommendation： nothing.

Knownsec

5. Appendix A：Contract code

Source code：

Youswap.sol

```
/**
 *Submitted for verification at Etherscan.io on 2021-03-26
*/

/**
 *Submitted for verification at Etherscan.io on 2021-03-26
*/

// File: @openzeppelin/contracts/utils/Address.sol

// SPDX-License-Identifier: MIT

pragma solidity >=0.6.2 <0.8.0;

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
```

Knownsec

```
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies on extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        uint256 size;
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
```

Knownsec

```
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding-and-decoding-functions[`abi.decode`].
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }
```

Knownsec

```
    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
     * `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data, string memory errorMessage) internal
returns (bytes memory) {
        return functionCallWithValue(target, data, 0, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal
returns (bytes memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call with value
failed");
    }

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-
uint256-}[`functionCallWithValue`], but
     * with `errorMessage` as a fallback revert reason when `target` reverts.
```

Knownsec

```
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: value }(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data) internal view returns (bytes memory) {
        return functionStaticCall(target, data, "Address: low-level static call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data, string memory errorMessage) internal view returns (bytes memory) {
        require(isContract(target), "Address: static call to non-contract");
```

Knownsec

```
        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.staticcall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory)
{
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data, string memory errorMessage)
internal returns (bytes memory) {
        require(isContract(target), "Address: delegate call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.delegatecall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }
```

Knownsec

```
    function _verifyCallResult(bool success, bytes memory returndata, string memory errorMessage)
private pure returns(bytes memory) {
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}

// File: @openzeppelin/contracts/math/SafeMath.sol



pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
```

Knownsec

```
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        if (b > a) return (false, 0);
        return (true, a - b);
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
```

Knownsec

```
 */
function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
    // benefit is lost if 'b' is also tested.
    // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
    if (a == 0) return (true, 0);
    uint256 c = a * b;
    if (c / a != b) return (false, 0);
    return (true, c);
}

/**
 * @dev Returns the division of two unsigned integers, with a division by zero flag.
 *
 * _Available since v3.4._
 */
function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    if (b == 0) return (false, 0);
    return (true, a / b);
}

/**
 * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
 *
 * _Available since v3.4._
 */
function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    if (b == 0) return (false, 0);
    return (true, a % b);
}

/**
 * @dev Returns the addition of two unsigned integers, reverting on
```

Knownsec

```
 * overflow.
 *
 * Counterpart to Solidity's `+` operator.
 *
 * Requirements:
 *
 * - Addition cannot overflow.
 */
function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    require(c >= a, "SafeMath: addition overflow");
    return c;
}

/**
 * @dev Returns the subtraction of two unsigned integers, reverting on
 * overflow (when the result is negative).
 *
 * Counterpart to Solidity's `-` operator.
 *
 * Requirements:
 *
 * - Subtraction cannot overflow.
 */
function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    require(b <= a, "SafeMath: subtraction overflow");
    return a - b;
}

/**
 * @dev Returns the multiplication of two unsigned integers, reverting on
 * overflow.
 *
```

Knownsec

```
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) return 0;
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting when dividing by zero.
```

Knownsec

```
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: modulo by zero");
        return a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        return a - b;
    }

    /**
```

Knownsec

```
     * @dev Returns the integer division of two unsigned integers, reverting with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryDiv}.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
```

Knownsec

```
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        return a % b;
    }
}

// File: @openzeppelin/contracts/token/ERC20/IERC20.sol



pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
```

Knownsec

```
 */
function transfer(address recipient, uint256 amount) external returns (bool);

/**
 * @dev Returns the remaining number of tokens that `spender` will be
 * allowed to spend on behalf of `owner` through {transferFrom}. This is
 * zero by default.
 *
 * This value changes when {approve} or {transferFrom} are called.
 */
function allowance(address owner, address spender) external view returns (uint256);

/**
 * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * IMPORTANT: Beware that changing an allowance with this method brings the risk
 * that someone may use both the old and the new allowance by unfortunate
 * transaction ordering. One possible solution to mitigate this race
 * condition is to first reduce the spender's allowance to 0 and set the
 * desired value afterwards:
 * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
 *
 * Emits an {Approval} event.
 */
function approve(address spender, uint256 amount) external returns (bool);

/**
 * @dev Moves `amount` tokens from `sender` to `recipient` using the
 * allowance mechanism. `amount` is then deducted from the caller's
 * allowance.
 *
```

Knownsec

```
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

// File: @openzeppelin/contracts/token/ERC20/SafeERC20.sol



pragma solidity >=0.6.0 <0.8.0;







/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
```

Knownsec

```
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for IERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to,
value));
    }

    /**
     * @dev Deprecated. This function has issues similar to the ones found in
     * {IERC20-approve}, and its usage is discouraged.
     *
     * Whenever possible, use {safeIncreaseAllowance} and
     * {safeDecreaseAllowance} instead.
     */
    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
```

Knownsec

```
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender,
newAllowance));
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20:
decreased allowance below zero");
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender,
newAllowance));
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxing the
requirement
     * on the return value: the return value is optional (but if data is returned, it must not be false).
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function _callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking
mechanism, since
        // we're implementing it ourselves. We use {Address.functionCall} to perform this call,
which verifies that
        // the target address contains contract code and also asserts for success in the low-level call.

        bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call
failed");
```

Knownsec

```
            if (returndata.length > 0) { // Return data is optional
                // solhint-disable-next-line max-line-length
                require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not
succeed");
            }
        }
}

// File: localhost/contract/library/ErrorCode.sol



pragma solidity 0.7.4;

library ErrorCode {

    string constant FORBIDDEN = 'YouSwap:FORBIDDEN';
    string constant IDENTICAL_ADDRESSES = 'YouSwap:IDENTICAL_ADDRESSES';
    string constant ZERO_ADDRESS = 'YouSwap:ZERO_ADDRESS';
    string constant INVALID_ADDRESSES = 'YouSwap:INVALID_ADDRESSES';
    string constant BALANCE_INSUFFICIENT = 'YouSwap:BALANCE_INSUFFICIENT';
    string constant REWARDTOTAL_LESS_THAN_REWARDPROVIDE =
'YouSwap:REWARDTOTAL_LESS_THAN_REWARDPROVIDE';
    string constant PARAMETER_TOO_LONG = 'YouSwap:PARAMETER_TOO_LONG';
    string constant REGISTERED = 'YouSwap:REGISTERED';
    string constant MINING_NOT_STARTED = 'YouSwap:MINING_NOT_STARTED';
    string constant END_OF_MINING = 'YouSwap:END_OF_MINING';
    string constant POOL_NOT_EXIST_OR_END_OF_MINING =
'YouSwap:POOL_NOT_EXIST_OR_END_OF_MINING';

}
// File: localhost/contract/interface/IYouswapInviteV1.sol
```

Knownsec

```
pragma solidity 0.7.4;

interface IYouswapInviteV1 {

    struct UserInfo {
        address upper;//上级
        address[] lowers;//下级
        uint256 startBlock;//邀请块高
    }

    event InviteV1(address indexed owner, address indexed upper, uint256 indexed height);//被邀请人的地址，邀请人的地址，邀请块高

    function inviteCount() external view returns (uint256);//邀请人数

    function inviteUpper1(address) external view returns (address);//上级邀请

    function inviteUpper2(address) external view returns (address, address);//上级邀请

    function inviteLower1(address) external view returns (address[] memory);//下级邀请

    function inviteLower2(address) external view returns (address[] memory, address[] memory);//下级邀请

    function inviteLower2Count(address) external view returns (uint256, uint256);//下级邀请

    function register() external returns (bool);//注册邀请关系

    function acceptInvitation(address) external returns (bool);//注册邀请关系
```

Knownsec

```
    function inviteBatch(address[] memory) external returns (uint, uint);//注册邀请关系：输入数量，成功数量

}
// File: localhost/contract/implement/YouswapInviteV1.sol



pragma solidity 0.7.4;



contract YouswapInviteV1 is IYouswapInviteV1 {

    address public constant ZERO = address(0);
    uint256 public startBlock;
    address[] public inviteUserInfoV1;
    mapping(address => UserInfo) public inviteUserInfoV2;

    constructor () {
        startBlock = block.number;
    }

    function inviteCount() override external view returns (uint256) {
        return inviteUserInfoV1.length;
    }

    function inviteUpper1(address _owner) override external view returns (address) {
        return inviteUserInfoV2[_owner].upper;
    }

    function inviteUpper2(address _owner) override external view returns (address, address) {
        address upper1 = inviteUserInfoV2[_owner].upper;
```

Knownsec

```
        address upper2 = address(0);
        if (address(0) != upper1) {
            upper2 = inviteUserInfoV2[upper1].upper;
        }

        return (upper1, upper2);
    }

    function inviteLower1(address _owner) override external view returns (address[] memory) {
        return inviteUserInfoV2[_owner].lowers;
    }

    function inviteLower2(address _owner) override external view returns (address[] memory,
address[] memory) {
        address[] memory lowers1 = inviteUserInfoV2[_owner].lowers;
        uint256 count = 0;
        uint256 lowers1Len = lowers1.length;
        for (uint256 i = 0; i < lowers1Len; i++) {
            count += inviteUserInfoV2[lowers1[i]].lowers.length;
        }
        address[] memory lowers;
        address[] memory lowers2 = new address[](count);
        count = 0;
        for (uint256 i = 0; i < lowers1Len; i++) {
            lowers = inviteUserInfoV2[lowers1[i]].lowers;
            for (uint256 j = 0; j < lowers.length; j++) {
                lowers2[count] = lowers[j];
                count++;
            }
        }

        return (lowers1, lowers2);
    }
```

Knownsec

```
function inviteLower2Count(address _owner) override external view returns (uint256, uint256) {
    address[] memory lowers1 = inviteUserInfoV2[_owner].lowers;
    uint256 lowers2Len = 0;
    uint256 len = lowers1.length;
    for (uint256 i = 0; i < len; i++) {
        lowers2Len += inviteUserInfoV2[lowers1[i]].lowers.length;
    }

    return (lowers1.length, lowers2Len);
}

function register() override external returns (bool) {
    UserInfo storage user = inviteUserInfoV2[tx.origin];
    require(0 == user.startBlock, ErrorCode.REGISTERED);
    user.upper = ZERO;
    user.startBlock = block.number;
    inviteUserInfoV1.push(tx.origin);

    emit InviteV1(tx.origin, user.upper, user.startBlock);

    return true;
}

function acceptInvitation(address _inviter) override external returns (bool) {
    require(msg.sender != _inviter, ErrorCode.FORBIDDEN);
    UserInfo storage user = inviteUserInfoV2[msg.sender];
    require(0 == user.startBlock, ErrorCode.REGISTERED);
    UserInfo storage upper = inviteUserInfoV2[_inviter];
    if (0 == upper.startBlock) {
        upper.upper = ZERO;
        upper.startBlock = block.number;
        inviteUserInfoV1.push(_inviter);
```

Knownsec

```
            emit InviteV1(_inviter, upper.upper, upper.startBlock);
        }
        user.upper = _inviter;
        upper.lowers.push(msg.sender);
        user.startBlock = block.number;
        inviteUserInfoV1.push(msg.sender);

        emit InviteV1(msg.sender, user.upper, user.startBlock);

        return true;
    }

    function inviteBatch(address[] memory _invitees) override external returns (uint, uint) {
        uint len = _invitees.length;
        require(len <= 100, ErrorCode.PARAMETER_TOO_LONG);
        UserInfo storage user = inviteUserInfoV2[msg.sender];
        if (0 == user.startBlock) {
            user.upper = ZERO;
            user.startBlock = block.number;
            inviteUserInfoV1.push(msg.sender);

            emit InviteV1(msg.sender, user.upper, user.startBlock);
        }
        uint count = 0;
        for (uint i = 0; i < len; i++) {
            if ((address(0) != _invitees[i]) && (msg.sender != _invitees[i])) {
                UserInfo storage lower = inviteUserInfoV2[_invitees[i]];
                if (0 == lower.startBlock) {
                    lower.upper = msg.sender;
                    lower.startBlock = block.number;
                    user.lowers.push(_invitees[i]);
                    inviteUserInfoV1.push(_invitees[i]);
```

Knownsec

```
                        count++;

                        emit InviteV1(_invitees[i], msg.sender, lower.startBlock);
                    }
                }
            }

            return (len, count);
        }

}
// File: localhost/contract/interface/ITokenYou.sol



pragma solidity 0.7.4;

interface ITokenYou {

    function mint(address recipient, uint256 amount) external;

    function decimals() external view returns (uint8);

}

// File: localhost/contract/interface/IYouswapFactoryV1.sol



pragma solidity 0.7.4;
```

Knownsec

```
/**
挖矿
 */
interface IYouswapFactoryV1 {

    /**
    用户挖矿信息
     */
    struct RewardInfo {
        uint256 receiveReward;//总领取奖励
        uint256 inviteReward;//总邀请奖励
        uint256 pledgeReward;//总质押奖励
    }

    /**
    质押用户信息
     */
    struct UserInfo {
        uint256 startBlock;//质押开始块高
        uint256 amount;//质押数量
        uint256 invitePower;//邀请算力
        uint256 pledgePower;//质押算力
        uint256 pendingReward;//待领取奖励
        uint256 inviteRewardDebt;//邀请负债
        uint256 pledgeRewardDebt;//质押负债
    }

    /**
    矿池信息（可视化）
     */
    struct PoolViewInfo {
        address lp;//LP 地址
        string name;//名称
```

Knownsec

```
        uint256 multiple;//奖励倍数
        uint256 priority;//排序
    }

    /**
    矿池信息
     */
    struct PoolInfo {
        uint256 startBlock;//挖矿开始块高
        uint256 rewardTotal;//矿池总奖励
        uint256 rewardProvide;//矿池已发放奖励
        address lp;//lp 合约地址
        uint256 amount;//质押数量
        uint256 lastRewardBlock;//最后发放奖励块高
        uint256 rewardPerBlock;//单个区块奖励
        uint256 totalPower;//总算力
        uint256 endBlock;//挖矿结束块高
        uint256 rewardPerShare;//单位算力奖励
    }

    ////////////////////////////////////////////////////////////////////////////

    /**
    自邀请
    self：Sender 地址
     */
    event InviteRegister(address indexed self);

    /**
    更新矿池信息

    action：true(新建矿池)，false(更新矿池)
    pool：矿池序号
```

Knownsec

lp：lp 合约地址

name：矿池名称

startBlock：矿池开始挖矿块高

rewardTotal：矿池总奖励

rewardPerBlock：区块奖励

multiple：矿池奖励倍数

priority：矿池排序

*/

event UpdatePool(bool action, uint256 pool, address indexed lp, string name, uint256 startBlock, uint256 rewardTotal, uint256 rewardPerBlock, uint256 multiple, uint256 priority);

/**

矿池挖矿结束

pool：矿池序号

lp：lp 合约地址

*/

event EndPool(uint256 pool, address indexed lp);

/**

质押

pool：矿池序号

lp：lp 合约地址

from：质押转出地址

amount：质押数量

*/

event Stake(uint256 pool, address indexed lp, address indexed from, uint256 amount);

/**

pool：矿池序号

lp：lp 合约地址

totalPower：矿池总算力

knownsec

owner：用户地址

ownerInvitePower：用户邀请算力

ownerPledgePower：用户质押算力

upper1：上 1 级地址

upper1InvitePower：上 1 级邀请算力

upper2：上 2 级地址

upper2InvitePower：上 2 级邀请算力

*/

event UpdatePower(uint256 pool, address lp, uint256 totalPower, address indexed owner, uint256 ownerInvitePower, uint256 ownerPledgePower, address indexed upper1, uint256 upper1InvitePower, address indexed upper2, uint256 upper2InvitePower);

//算力

/**

解质押

pool：矿池序号

lp：lp 合约地址

to：解质押转入地址

amount：解质押数量

*/

event UnStake(uint256 pool, address indexed lp, address indexed to, uint256 amount);

/**

提取奖励

pool：矿池序号

lp：lp 合约地址

to：奖励转入地址

amount：奖励数量

*/

event WithdrawReward(uint256 pool, address indexed lp, address indexed to, uint256 amount);

Knownsec

```
/**
挖矿

pool：矿池序号
lp：lp 合约地址
amount：奖励数量
 */
event Mint(uint256 pool, address indexed lp, uint256 amount);

////////////////////////////////////////////////////////////////////////////////

/**
修改 OWNER
 */
function transferOwnership(address) external;

/**
设置 YOU
 */
function setYou(ITokenYou) external;

/**
设置邀请关系
 */
function setInvite(YouswapInviteV1) external;

/**
质押
*/
function deposit(uint256, uint256) external;

/**
```

Knownsec

```
解质押、提取奖励
 */
function withdraw(uint256, uint256) external;

/**
矿池质押地址
 */
function poolPledgeAddresss(uint256) external view returns (address[] memory);

/**
算力占比
 */
function powerScale(uint256, address) external view returns (uint256);

/**
待领取的奖励
 */
function pendingReward(uint256, address) external view returns (uint256);

/**
下级收益贡献
 */
function rewardContribute(address, address) external view returns (uint256);

/**
个人收益加成
 */
function selfReward(address) external view returns (uint256);

/**
通过 lp 查询矿池编号
 */
function poolNumbers(address) external view returns (uint256[] memory);
```

Knownsec

```
/**
设置运营权限
 */
function setOperateOwner(address, bool) external;

////////////////////////////////////////////////////////////////////////////////

/**
新建矿池
 */
function addPool(string memory, address, uint256, uint256) external returns (bool);

/**
修改矿池区块奖励
 */
function setRewardPerBlock(uint256, uint256) external;

/**
修改矿池总奖励
 */
function setRewardTotal(uint256, uint256) external;

/**
修改矿池名称
 */
function setName(uint256, string memory) external;

/**
修改矿池倍数
 */
function setMultiple(uint256, uint256) external;
```

Knownsec

```
    /**
    修改矿池排序
     */
    function setPriority(uint256, uint256) external;

    ////////////////////////////////////////////////////////////////////////////////

}
// File: localhost/contract/implement/YouswapFactoryV1.sol



pragma solidity 0.7.4;




contract YouswapFactoryV1 is IYouswapFactoryV1 {

    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    uint256 public deployBlock;//合约部署块高
    address public owner;//所有权限
    mapping(address => bool) public operateOwner;//运营权限
    ITokenYou public you;//you contract
    YouswapInviteV1 public invite;//invite contract

    uint256 public poolCount = 0;//矿池数量
    mapping(address => RewardInfo) public rewardInfos;//用户挖矿信息
    mapping(uint256 => PoolInfo) public poolInfos;//矿池信息
    mapping(uint256 => PoolViewInfo) public poolViewInfos;//矿池信息
```

Knownsec

```
    mapping(uint256 => address[]) public pledgeAddresss;//矿池质押地址
    mapping(uint256 => mapping(address => UserInfo)) public pledgeUserInfo;//矿池质押用户信息

    uint256 public constant inviteSelfReward = 5;//质押自奖励，5%
    uint256 public constant invite1Reward = 15;//1 级邀请奖励，15%
    uint256 public constant invite2Reward = 10;//2 级邀请奖励，10%
    uint256 public constant rewardPerBlock = 267094;//区块奖励
    uint256 public rewardTotal = 0;//总挖矿奖励

    constructor (ITokenYou _you, YouswapInviteV1 _invite) {
        deployBlock = block.number;
        owner = msg.sender;
        invite = _invite;
        _setOperateOwner(owner, true);
        _setYou(_you);
    }

    ////////////////////////////////////////////////////////////////////////////////

    function transferOwnership(address _owner) override external {
        require(owner == msg.sender, ErrorCode.FORBIDDEN);
        require((address(0) != _owner) && (owner != _owner),
ErrorCode.INVALID_ADDRESSES);
        _setOperateOwner(owner, false);
        _setOperateOwner(_owner, true);
        owner = _owner;
    }

    function setYou(ITokenYou _you) override external {
        _setYou(_you);
    }

    function _setYou(ITokenYou _you) internal {
```

Knownsec

```
        require(owner == msg.sender, ErrorCode.FORBIDDEN);
        you = _you;
    }

    function setInvite(YouswapInviteV1 _invite) override external {
        require(owner == msg.sender, ErrorCode.FORBIDDEN);
        invite = _invite;
    }

    function deposit(uint256 _pool, uint256 _amount) override external {
        require(0 < _amount, ErrorCode.FORBIDDEN);
        PoolInfo storage poolInfo = poolInfos[_pool];
        require((address(0) != poolInfo.lp) && (poolInfo.startBlock <= block.number),
ErrorCode.MINING_NOT_STARTED);
        //require(0 == poolInfo.endBlock, ErrorCode.END_OF_MINING);
        (, uint256 startBlock) = invite.inviteUserInfoV2(msg.sender);
        if (0 == startBlock) {
            invite.register();

            emit InviteRegister(msg.sender);
        }

        IERC20(poolInfo.lp).safeTransferFrom(msg.sender, address(this), _amount);

        (address upper1, address upper2) = invite.inviteUpper2(msg.sender);

        computeReward(_pool);

        provideReward(_pool, poolInfo.rewardPerShare, poolInfo.lp, msg.sender, upper1, upper2);

        addPower(_pool, msg.sender, _amount, upper1, upper2);

        setRewardDebt(_pool, poolInfo.rewardPerShare, msg.sender, upper1, upper2);
```

Knownsec

```
        emit Stake(_pool, poolInfo.lp, msg.sender, _amount);
    }

    function withdraw(uint256 _pool, uint256 _amount) override external {
        PoolInfo storage poolInfo = poolInfos[_pool];
        require((address(0) != poolInfo.lp) && (poolInfo.startBlock <= block.number),
ErrorCode.MINING_NOT_STARTED);
        if (0 < _amount) {
            UserInfo storage userInfo = pledgeUserInfo[_pool][msg.sender];
            require(_amount <= userInfo.amount, ErrorCode.BALANCE_INSUFFICIENT);
            IERC20(poolInfo.lp).safeTransfer(msg.sender, _amount);

            emit UnStake(_pool, poolInfo.lp, msg.sender, _amount);
        }

        (address _upper1, address _upper2) = invite.inviteUpper2(msg.sender);

        computeReward(_pool);

        provideReward(_pool, poolInfo.rewardPerShare, poolInfo.lp, msg.sender, _upper1,
_upper2);

        if (0 < _amount) {
            subPower(_pool, msg.sender, _amount, _upper1, _upper2);
        }

        setRewardDebt(_pool, poolInfo.rewardPerShare, msg.sender, _upper1, _upper2);
    }

    function poolPledgeAddresss(uint256 _pool) override external view returns (address[] memory) {
        return pledgeAddresss[_pool];
    }
```

Knownsec

```
    function computeReward(uint256 _pool) internal {
        PoolInfo storage poolInfo = poolInfos[_pool];
        if ((0 < poolInfo.totalPower) && (poolInfo.rewardProvide < poolInfo.rewardTotal)) {
            uint256 reward = (block.number -
poolInfo.lastRewardBlock).mul(poolInfo.rewardPerBlock);
            if (poolInfo.rewardProvide.add(reward) > poolInfo.rewardTotal) {
                reward = poolInfo.rewardTotal.sub(poolInfo.rewardProvide);
                poolInfo.endBlock = block.number;
            }

            rewardTotal = rewardTotal.add(reward);
            poolInfo.rewardProvide = poolInfo.rewardProvide.add(reward);
            poolInfo.rewardPerShare =
poolInfo.rewardPerShare.add(reward.mul(1e24).div(poolInfo.totalPower));
            poolInfo.lastRewardBlock = block.number;

            emit Mint(_pool, poolInfo.lp, reward);

            if (0 < poolInfo.endBlock) {
                emit EndPool(_pool, poolInfo.lp);
            }
        }
    }

    function addPower(uint256 _pool, address _user, uint256 _amount, address _upper1, address
_upper2) internal {
        PoolInfo storage poolInfo = poolInfos[_pool];
        poolInfo.amount = poolInfo.amount.add(_amount);

        uint256 pledgePower = _amount;
        UserInfo storage userInfo = pledgeUserInfo[_pool][_user];
        userInfo.amount = userInfo.amount.add(_amount);
```

Knownsec

```
userInfo.pledgePower = userInfo.pledgePower.add(pledgePower);
poolInfo.totalPower = poolInfo.totalPower.add(pledgePower);
if (0 == userInfo.startBlock) {
    userInfo.startBlock = block.number;
    pledgeAddresss[_pool].push(msg.sender);
}

uint256 upper1InvitePower = 0;
uint256 upper2InvitePower = 0;

if (address(0) != _upper1) {
    uint256 inviteSelfPower = pledgePower.mul(inviteSelfReward).div(100);
    userInfo.invitePower = userInfo.invitePower.add(inviteSelfPower);
    poolInfo.totalPower = poolInfo.totalPower.add(inviteSelfPower);

    uint256 invite1Power = pledgePower.mul(invite1Reward).div(100);
    UserInfo storage upper1Info = pledgeUserInfo[_pool][_upper1];
    upper1Info.invitePower = upper1Info.invitePower.add(invite1Power);
    upper1InvitePower = upper1Info.invitePower;
    poolInfo.totalPower = poolInfo.totalPower.add(invite1Power);
    if (0 == upper1Info.startBlock) {
        upper1Info.startBlock = block.number;
        pledgeAddresss[_pool].push(_upper1);
    }
}

if (address(0) != _upper2) {
    uint256 invite2Power = pledgePower.mul(invite2Reward).div(100);
    UserInfo storage upper2Info = pledgeUserInfo[_pool][_upper2];
    upper2Info.invitePower = upper2Info.invitePower.add(invite2Power);
    upper2InvitePower = upper2Info.invitePower;
    poolInfo.totalPower = poolInfo.totalPower.add(invite2Power);
    if (0 == upper2Info.startBlock) {
```

Knownsec

```
                upper2Info.startBlock = block.number;
                pledgeAddresss[_pool].push(_upper2);
            }
        }

        emit UpdatePower(_pool, poolInfo.lp, poolInfo.totalPower, _user, userInfo.invitePower,
userInfo.pledgePower, _upper1, upper1InvitePower, _upper2, upper2InvitePower);
    }

    function subPower(uint256 _pool, address _user, uint256 _amount, address _upper1, address
_upper2) internal {
        PoolInfo storage poolInfo = poolInfos[_pool];
        UserInfo storage userInfo = pledgeUserInfo[_pool][_user];
        poolInfo.amount = poolInfo.amount.sub(_amount);

        uint256 pledgePower = _amount;
        userInfo.amount = userInfo.amount.sub(_amount);
        userInfo.pledgePower = userInfo.pledgePower.sub(pledgePower);
        poolInfo.totalPower = poolInfo.totalPower.sub(pledgePower);

        uint256 upper1InvitePower = 0;
        uint256 upper2InvitePower = 0;

        if (address(0) != _upper1) {
            uint256 inviteSelfPower = pledgePower.mul(inviteSelfReward).div(100);
            userInfo.invitePower = userInfo.invitePower.sub(inviteSelfPower);
            poolInfo.totalPower = poolInfo.totalPower.sub(inviteSelfPower);

            UserInfo storage upper1Info = pledgeUserInfo[_pool][_upper1];
            if (0 < upper1Info.startBlock) {
                uint256 invite1Power = pledgePower.mul(invite1Reward).div(100);
                upper1Info.invitePower = upper1Info.invitePower.sub(invite1Power);
                upper1InvitePower = upper1Info.invitePower;
```

Knownsec

```
                poolInfo.totalPower = poolInfo.totalPower.sub(invite1Power);

                if (address(0) != _upper2) {
                    UserInfo storage upper2Info = pledgeUserInfo[_pool][_upper2];
                    if (0 < upper2Info.startBlock) {
                        uint256 invite2Power = pledgePower.mul(invite2Reward).div(100);
                        upper2Info.invitePower = upper2Info.invitePower.sub(invite2Power);
                        upper2InvitePower = upper2Info.invitePower;
                        poolInfo.totalPower = poolInfo.totalPower.sub(invite2Power);
                    }
                }
            }
        }

        emit UpdatePower(_pool, poolInfo.lp, poolInfo.totalPower, _user, userInfo.invitePower,
userInfo.pledgePower, _upper1, upper1InvitePower, _upper2, upper2InvitePower);
    }

    function provideReward(uint256 _pool, uint256 _rewardPerShare, address _lp, address _user,
address _upper1, address _upper2) internal {
        uint256 inviteReward = 0;
        uint256 pledgeReward = 0;
        UserInfo storage userInfo = pledgeUserInfo[_pool][_user];
        if ((0 < userInfo.invitePower) || (0 < userInfo.pledgePower)) {
            inviteReward =
userInfo.invitePower.mul(_rewardPerShare).sub(userInfo.inviteRewardDebt).div(1e24);
            pledgeReward =
userInfo.pledgePower.mul(_rewardPerShare).sub(userInfo.pledgeRewardDebt).div(1e24);

            userInfo.pendingReward =
userInfo.pendingReward.add(inviteReward.add(pledgeReward));

            RewardInfo storage userRewardInfo = rewardInfos[_user];
```

Knownsec

```
            userRewardInfo.inviteReward = userRewardInfo.inviteReward.add(inviteReward);
            userRewardInfo.pledgeReward = userRewardInfo.pledgeReward.add(pledgeReward);
        }

        if (0 < userInfo.pendingReward) {
            you.mint(_user, userInfo.pendingReward);

            RewardInfo storage userRewardInfo = rewardInfos[_user];
            userRewardInfo.receiveReward = userRewardInfo.inviteReward;

            emit WithdrawReward(_pool, _lp, _user, userInfo.pendingReward);

            userInfo.pendingReward = 0;
        }

        if (address(0) != _upper1) {
            UserInfo storage upper1Info = pledgeUserInfo[_pool][_upper1];
            if ((0 < upper1Info.invitePower) || (0 < upper1Info.pledgePower)) {
                inviteReward =
upper1Info.invitePower.mul(_rewardPerShare).sub(upper1Info.inviteRewardDebt).div(1e24);
                pledgeReward =
upper1Info.pledgePower.mul(_rewardPerShare).sub(upper1Info.pledgeRewardDebt).div(1e24);

                upper1Info.pendingReward =
upper1Info.pendingReward.add(inviteReward.add(pledgeReward));

                RewardInfo storage upper1RewardInfo = rewardInfos[_upper1];
                upper1RewardInfo.inviteReward =
upper1RewardInfo.inviteReward.add(inviteReward);
                upper1RewardInfo.pledgeReward =
upper1RewardInfo.pledgeReward.add(pledgeReward);
            }
```

Knownsec

```
                if (address(0) != _upper2) {
                    UserInfo storage upper2Info = pledgeUserInfo[_pool][_upper2];
                    if ((0 < upper2Info.invitePower) || (0 < upper2Info.pledgePower)) {
                        inviteReward =
upper2Info.invitePower.mul(_rewardPerShare).sub(upper2Info.inviteRewardDebt).div(1e24);
                        pledgeReward =
upper2Info.pledgePower.mul(_rewardPerShare).sub(upper2Info.pledgeRewardDebt).div(1e24);

                        upper2Info.pendingReward =
upper2Info.pendingReward.add(inviteReward.add(pledgeReward));

                        RewardInfo storage upper2RewardInfo = rewardInfos[_upper2];
                        upper2RewardInfo.inviteReward =
upper2RewardInfo.inviteReward.add(inviteReward);
                        upper2RewardInfo.pledgeReward =
upper2RewardInfo.pledgeReward.add(pledgeReward);
                    }
                }
            }
        }

    function setRewardDebt(uint256 _pool, uint256 _rewardPerShare, address _user, address
_upper1, address _upper2) internal {
        UserInfo storage userInfo = pledgeUserInfo[_pool][_user];
        userInfo.inviteRewardDebt = userInfo.invitePower.mul(_rewardPerShare);
        userInfo.pledgeRewardDebt = userInfo.pledgePower.mul(_rewardPerShare);

        if (address(0) != _upper1) {
            UserInfo storage upper1Info = pledgeUserInfo[_pool][_upper1];
            upper1Info.inviteRewardDebt = upper1Info.invitePower.mul(_rewardPerShare);
            upper1Info.pledgeRewardDebt = upper1Info.pledgePower.mul(_rewardPerShare);

            if (address(0) != _upper2) {
```

Knownsec

```
                UserInfo storage upper2Info = pledgeUserInfo[_pool][_upper2];
                upper2Info.inviteRewardDebt = upper2Info.invitePower.mul(_rewardPerShare);
                upper2Info.pledgeRewardDebt = upper2Info.pledgePower.mul(_rewardPerShare);
            }
        }
    }

    function powerScale(uint256 _pool, address _user) override external view returns (uint256) {
        PoolInfo memory poolInfo = poolInfos[_pool];
        if (0 == poolInfo.totalPower) {
            return 0;
        }

        UserInfo memory userInfo = pledgeUserInfo[_pool][_user];
        return (userInfo.invitePower.add(userInfo.pledgePower).mul(100)).div(poolInfo.totalPower);
    }

    function pendingReward(uint256 _pool, address _user) override external view returns (uint256) {
        uint256 totalReward = 0;
        PoolInfo memory poolInfo = poolInfos[_pool];
        if (address(0) != poolInfo.lp && (poolInfo.startBlock <= block.number)) {
            uint256 rewardPerShare = 0;
            if (0 < poolInfo.totalPower) {
                uint256 reward = (block.number -
poolInfo.lastRewardBlock).mul(poolInfo.rewardPerBlock);
                if (poolInfo.rewardProvide.add(reward) > poolInfo.rewardTotal) {
                    reward = poolInfo.rewardTotal.sub(poolInfo.rewardProvide);
                }
                rewardPerShare = reward.mul(1e24).div(poolInfo.totalPower);
            }
            rewardPerShare = rewardPerShare.add(poolInfo.rewardPerShare);

            UserInfo memory userInfo = pledgeUserInfo[_pool][_user];
```

Knownsec

```
                totalReward = userInfo.pendingReward;
                totalReward =
totalReward.add(userInfo.invitePower.mul(rewardPerShare).sub(userInfo.inviteRewardDebt).div(1e24
));
                totalReward =
totalReward.add(userInfo.pledgePower.mul(rewardPerShare).sub(userInfo.pledgeRewardDebt).div(1e2
4));
            }

            return totalReward;
        }

        function rewardContribute(address _user, address _lower) override external view returns (uint256)
{
            if ((address(0) == _user) || (address(0) == _lower)) {
                return 0;
            }

            uint256 inviteReward = 0;
            (address upper1, address upper2) = invite.inviteUpper2(_lower);
            if (_user == upper1) {
                inviteReward = rewardInfos[_lower].pledgeReward.mul(invite1Reward).div(100);
            }else if (_user == upper2) {
                inviteReward = rewardInfos[_lower].pledgeReward.mul(invite2Reward).div(100);
            }

            return inviteReward;
        }

        function selfReward(address _user) override external view returns (uint256) {
            address upper1 = invite.inviteUpper1(_user);
            if (address(0) == upper1) {
                return 0;
```

Knownsec

```
        }

        RewardInfo memory userRewardInfo = rewardInfos[_user];
        return userRewardInfo.pledgeReward.mul(inviteSelfReward).div(100);
    }

    function poolNumbers(address _lp) override external view returns (uint256[] memory) {
        uint256 count = 0;
        for (uint256 i = 0; i < poolCount; i++) {
            if (_lp == poolViewInfos[i].lp) {
                count = count.add(1);
            }
        }

        uint256[] memory numbers = new uint256[](count);
        count = 0;
        for (uint256 i = 0; i < poolCount; i++) {
            if (_lp == poolViewInfos[i].lp) {
                numbers[count] = i;
                count = count.add(1);
            }
        }

        return numbers;
    }

    function setOperateOwner(address _address, bool _bool) override external {
        _setOperateOwner(_address, _bool);
    }

    function _setOperateOwner(address _address, bool _bool) internal {
        require(owner == msg.sender, ErrorCode.FORBIDDEN);
        operateOwner[_address] = _bool;
```

Knownsec

```
    }

    ////////////////////////////////////////////////////////////////////////////////

    function addPool(string memory _name, address _lp, uint256 _startBlock, uint256 _rewardTotal)
override external returns (bool) {
        require(operateOwner[msg.sender] && (address(0) != _lp) && (address(this) != _lp),
ErrorCode.FORBIDDEN);
        _startBlock = _startBlock < block.number ? block.number : _startBlock;
        uint256 _pool = poolCount;
        poolCount = poolCount.add(1);

        PoolViewInfo storage poolViewInfo = poolViewInfos[_pool];
        poolViewInfo.lp = _lp;
        poolViewInfo.name = _name;
        poolViewInfo.multiple = 1;
        poolViewInfo.priority = _pool.mul(100);

        PoolInfo storage poolInfo = poolInfos[_pool];
        poolInfo.startBlock = _startBlock;
        poolInfo.rewardTotal = _rewardTotal;
        poolInfo.rewardProvide = 0;
        poolInfo.lp = _lp;
        poolInfo.amount = 0;
        poolInfo.lastRewardBlock = _startBlock.sub(1);
        poolInfo.rewardPerBlock = rewardPerBlock;
        poolInfo.totalPower = 0;
        poolInfo.endBlock = 0;
        poolInfo.rewardPerShare = 0;

        emit UpdatePool(true, _pool, poolInfo.lp, poolViewInfo.name, poolInfo.startBlock,
poolInfo.rewardTotal, poolInfo.rewardPerBlock, poolViewInfo.multiple, poolViewInfo.priority);
```

Knownsec

```
        return true;
    }

    function setRewardPerBlock(uint256 _pool, uint256 _rewardPerBlock) override external {
        require(operateOwner[msg.sender], ErrorCode.FORBIDDEN);
        PoolInfo storage poolInfo = poolInfos[_pool];
        require((address(0) != poolInfo.lp) && (0 == poolInfo.endBlock),
ErrorCode.POOL_NOT_EXIST_OR_END_OF_MINING);
        poolInfo.rewardPerBlock = _rewardPerBlock;

        PoolViewInfo memory poolViewInfo = poolViewInfos[_pool];

        emit UpdatePool(false, _pool, poolInfo.lp, poolViewInfo.name, poolInfo.startBlock,
poolInfo.rewardTotal, poolInfo.rewardPerBlock, poolViewInfo.multiple, poolViewInfo.priority);
    }

    function setRewardTotal(uint256 _pool, uint256 _rewardTotal) override external {
        require(operateOwner[msg.sender], ErrorCode.FORBIDDEN);
        PoolInfo storage poolInfo = poolInfos[_pool];
        require((address(0) != poolInfo.lp) && (0 == poolInfo.endBlock),
ErrorCode.POOL_NOT_EXIST_OR_END_OF_MINING);
        require(poolInfo.rewardProvide < _rewardTotal,
ErrorCode.REWARDTOTAL_LESS_THAN_REWARDPROVIDE);
        poolInfo.rewardTotal = _rewardTotal;

        PoolViewInfo memory poolViewInfo = poolViewInfos[_pool];

        emit UpdatePool(false, _pool, poolInfo.lp, poolViewInfo.name, poolInfo.startBlock,
poolInfo.rewardTotal, poolInfo.rewardPerBlock, poolViewInfo.multiple, poolViewInfo.priority);
    }

    function setName(uint256 _pool, string memory _name) override external {
        require(operateOwner[msg.sender], ErrorCode.FORBIDDEN);
```

Knownsec

```
        PoolViewInfo storage poolViewInfo = poolViewInfos[_pool];
        require(address(0) != poolViewInfo.lp,
ErrorCode.POOL_NOT_EXIST_OR_END_OF_MINING);
        poolViewInfo.name = _name;

        PoolInfo memory poolInfo = poolInfos[_pool];

        emit UpdatePool(false, _pool, poolInfo.lp, poolViewInfo.name, poolInfo.startBlock,
poolInfo.rewardTotal, poolInfo.rewardPerBlock, poolViewInfo.multiple, poolViewInfo.priority);
    }

    function setMultiple(uint256 _pool, uint256 _multiple) override external {
        require(operateOwner[msg.sender], ErrorCode.FORBIDDEN);
        PoolViewInfo storage poolViewInfo = poolViewInfos[_pool];
        require(address(0) != poolViewInfo.lp,
ErrorCode.POOL_NOT_EXIST_OR_END_OF_MINING);
        poolViewInfo.multiple = _multiple;

        PoolInfo memory poolInfo = poolInfos[_pool];

        emit UpdatePool(false, _pool, poolInfo.lp, poolViewInfo.name, poolInfo.startBlock,
poolInfo.rewardTotal, poolInfo.rewardPerBlock, poolViewInfo.multiple, poolViewInfo.priority);
    }

    function setPriority(uint256 _pool, uint256 _priority) override external {
        require(operateOwner[msg.sender], ErrorCode.FORBIDDEN);
        PoolViewInfo storage poolViewInfo = poolViewInfos[_pool];
        require(address(0) != poolViewInfo.lp,
ErrorCode.POOL_NOT_EXIST_OR_END_OF_MINING);
        poolViewInfo.priority = _priority;

        PoolInfo memory poolInfo = poolInfos[_pool];
```

Knownsec

```
        emit UpdatePool(false, _pool, poolInfo.lp, poolViewInfo.name, poolInfo.startBlock,
poolInfo.rewardTotal, poolInfo.rewardPerBlock, poolViewInfo.multiple, poolViewInfo.priority);
    }

    ////////////////////////////////////////////////////////////////////////////////

}
```

Knownsec

6. Appendix B: Vulnerability rating standard

Smart contract vulnerability rating standards	
Level	Level Description
High	Vulnerabilities that can directly cause the loss of token contracts or user funds, such as: value overflow loopholes that can cause the value of tokens to zero, fake recharge loopholes that can cause exchanges to lose tokens, and can cause contract accounts to lose ETH or tokens. Access loopholes, etc.; Vulnerabilities that can cause loss of ownership of token contracts, such as: access control defects of key functions, call injection leading to bypassing of access control of key functions, etc.; Vulnerabilities that can cause the token contract to not work properly, such as: denial of service vulnerability caused by sending ETH to malicious addresses, and denial of service vulnerability caused by exhaustion of gas.
Medium	High-risk vulnerabilities that require specific addresses to trigger, such as value overflow vulnerabilities that can be triggered by token contract owners; access control defects for non-critical functions, and logical design defects that cannot cause direct capital losses, etc.
Low	Vulnerabilities that are difficult to be triggered, vulnerabilities with limited damage after triggering, such as value overflow vulnerabilities that require a large amount of ETH or tokens to trigger, vulnerabilities where attackers cannot

Knownsec

	directly profit after triggering value overflow, and the transaction sequence triggered by specifying high gas depends on the risk Wait.

Knownsec

7. Appendix C：Introduction to auditing tools

7.1 Manticore

Manticore is a symbolic execution tool for analyzing binary files and smart contracts. Manticore includes a symbolic Ethereum Virtual Machine (EVM), an EVM disassembler/assembler and a convenient interface for automatic compilation and analysis of Solidity. It also integrates Ethersplay, Bit of Traits of Bits visual disassembler for EVM bytecode, used for visual analysis. Like binary files, Manticore provides a simple command line interface and a Python for analyzing EVM bytecode API.

7.2 Oyente

Oyente is a smart contract analysis tool. Oyente can be used to detect common bugs in smart contracts, such as reentrancy, transaction sequencing dependencies, etc. More convenient, Oyente's design is modular, so this allows advanced users to implement and Insert their own detection logic to check the custom attributes in their contract.

7.3 securify.sh

Securify can verify common security issues of Ethereum smart contracts, such as disordered transactions and lack of input verification. It analyzes all possible execution paths of the program while fully automated. In addition, Securify also has a

Knownsec

specific language for specifying vulnerabilities, which makes Securify can keep an eye on current security and other reliability issues at any time.

7.4 Echidna

Echidna is a Haskell library designed for fuzzing EVM code.

7.5 MAIAN

MAIAN is an automated tool for finding vulnerabilities in Ethereum smart contracts. Maian processes the bytecode of the contract and tries to establish a series of transactions to find and confirm the error.

7.6 ethersplay

ethersplay is an EVM disassembler, which contains relevant analysis tools.

7.7 ida-evm

ida-evm is an IDA processor module for the Ethereum Virtual Machine (EVM).

7.8 Remix-ide

ida-evm is an IDA processor module for the Ethereum Virtual Machine (EVM).

knownsec

7.9 Knownsec Penetration Tester Special Toolkit

Pen-Tester tools collection is created by KnownSec team. It contains plenty of Pen-Testing tools such as automatic testing tool, scripting tool, Self-developed tools etc.

Knownsec

KNOWNSEC

Beijing KnownSec Information Technology Co., Ltd.

Advisory telephone	+86(10)400 060 9587
E-mail	sec@knownsec.com
Website	www.knownsec.com
Address	wangjing soho T2-B2509,Chaoyang District, Beijing